Dynamic MIPS Rate Stabilization in Out-of-Order Processors

Jinho Suh Michel Dubois

Ming Hsieh Department of Electrical Engineering
Viterbi School of Engineering
University of Southern California
Los Angeles, CA 90089-2560

Technical Report # CENG-2009-1

January 2009

Dynamic MIPS Rate Stabilization in Out-of-Order Processors

Jinho Suh and Michel Dubois

Department of Computer Engineering
University of Southern California
Los Angeles, CA 90089-2560
jinhosuh@usc.edu, dubois@paris.usc.edu

Abstract

Today's micro-processor cores reach high performance levels not only by their high clock rate but also by the concurrent execution of a large number of instructions. Because of the relationship between power and frequency, it becomes attractive to run an OoO (Out-of-Order) processor at a frequency lower than its nominal frequency in the context of embedded or real-time systems. Unfortunately, whereas OoO processors have high average throughput, their highly variable and hard-to-predict execution rate make them unsuitable for real-time systems with hard or even soft deadlines.

In this paper, we show that the throughput of an OoO processor can be stable and predictable by controlling its MIPS (Mega Instructions Per Second) rate via a PID (Proportional, Integral, and Differential gain) feedback controller and DVFS (Dynamic Voltage and Frequency Scaling). The controller controls on-chip supply voltage (Vdd) and operating frequency in order to sustain a target MIPS rate. The stabilized processor uses much less power per committed instruction, because of its reduced average frequency. The EPI (Energy Per Instruction) is also lowered by an average of 28% across our benchmark programs. Since a stable MIPS rate is maintained consistently and power/energy per instruction is reduced, OoO processors stabilized by a feedback controller can realistically be deployed in real-time systems with soft or hard deadlines.

To demonstrate this capability we select a subset of the MiBench benchmarks that display the widest execution rate variations and stabilize their MIPS rate in the context of a 1GHz Pentium III-like micro-architecture.

1. Introduction

Limiting power has become one of the most critical design constraints for portable or mobile platforms as well as for desktop or server platforms. Besides technology and circuit design, architecture has a significant role to play in reducing power dissipation for a given level of performance. DVFS (Dynamic Voltage and Frequency Scaling) schemes have been exploited in various ways in processor designs to improve power efficiency [5, 10, 11, 17, 20, 22, 24, 26, 27, 29, 31, 36, 38]. A number of control algorithms optimize power by tuning frequency and supply voltage, while meeting performance goals. DVFS is very effective at regulating power since dynamic power consumption, which remains the dominant factor in the power equation, is proportional to the cube of the frequency. Lowering the on-chip operating frequency together with on-chip Vdd by a factor k reduces power consumption by a factor k^3 [31]. Thus many processors for mobile applications implement a DVFS scheme nowadays.

Many mobile applications are real-time applications. The processor used in these applications must provide predictable turn-around time for the tasks it runs. Power efficiency is a critical factor as well. Performance and power often clash: in general higher performance requires more power. One way to achieve higher performance within a given power budget is to exploit parallelism at all levels. OoO (Out-of-Order) processors exploit ILP (Instruction Level Parallelism), and, when optimized for power efficiency, could be well suited for real-time embedded applications. Unfortunately, whereas OoO processors have

high average throughput, their execution rate is highly variable, program-dependent and unpredictable because their superior performance relies on speculation and cache hierarchies. This uncertainty makes it hard to estimate precisely their execution time with techniques such as WCET (Worst Case Execution Time) analysis, on which software developers for real-time systems rely. Software developers can never be sure that the execution of their application will not overrun deadlines.

In this paper, we consider a processor as a supplier of QoS (Quality of Service). We show how to stabilize the performance of an OoO processor at a predetermined, steady instruction throughput level with a PID feedback control approach, which is equivalent to maintaining a constant level of QoS. By exploiting parallelism instead of frequency to reach a given performance level, OoO processors stabilized by feedback control are power/energy efficient and their execution time becomes predictable as well.

The major contribution of this paper is to demonstrate that the throughput of an OoO processor (in term of the number of committed instructions per second) can be stabilized so that real-time applications can meet their deadline right on time and, at the same time, lower power and energy by minimizing the clock rate to meet the deadline. In this paper we provide answers to the following questions:

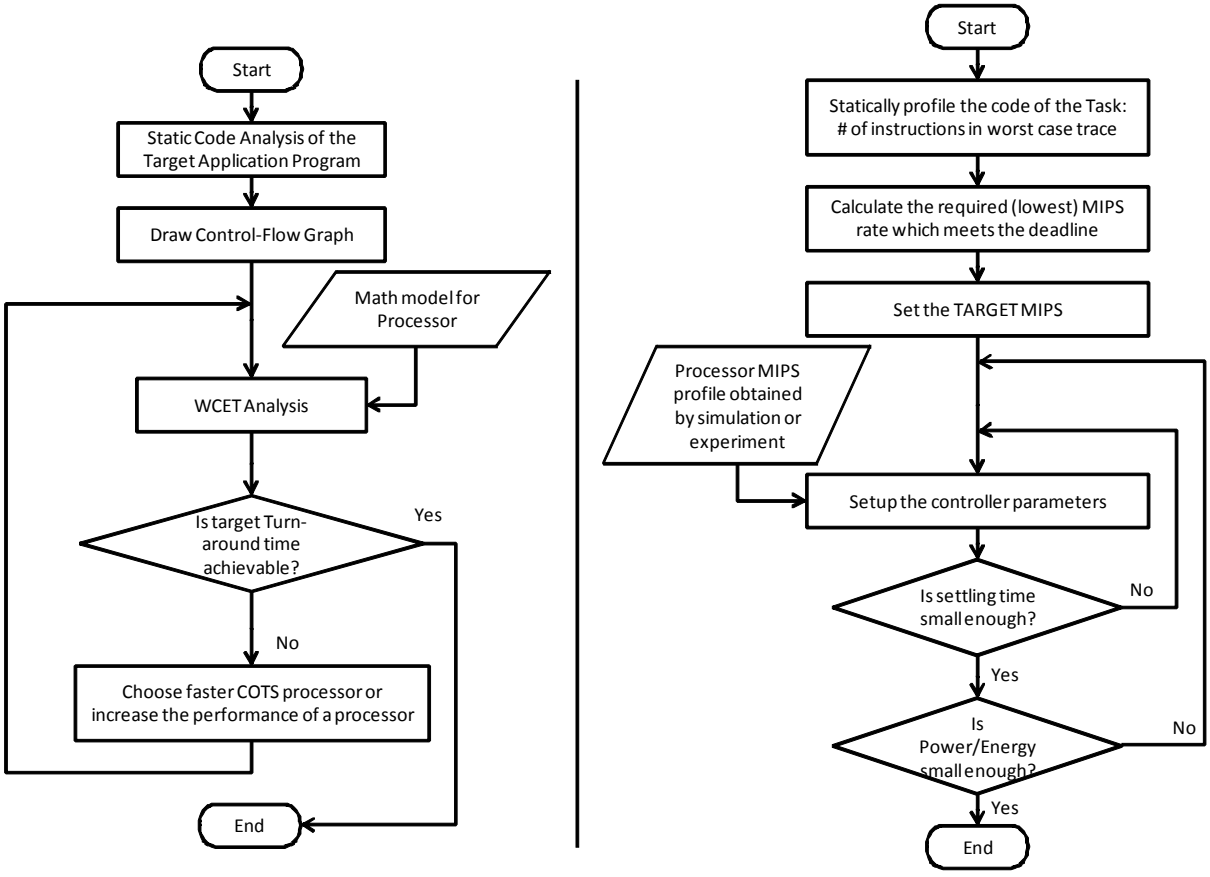
- Is performance stabilization needed? In other words, does the processing rate of OoO processors vary widely across programs and program phases in a way that affects application developers?
- What are the major advantages achieved by performance stabilization?
- Can a control algorithm to stabilize the processor's throughput be simple enough to be easily implemented in software or hardware?
- How do the values of the control parameters affect stabilization?
- How does the stabilized processor perform on actual program segments?
- How does a stabilized OoO processor compare with a traditional IO (in-order) processor in the context of real-time systems?

In Sections 2 and 3 we introduce our proposed framework for real-time application development and then review prior work on this topic. In Section 4, we evaluate the variability of instruction throughput in out-of-order processors, and demonstrate that stabilization is required for real-time systems with deadlines. In Section 5, we propose a PID feedback control approach integrated with a performance-to-frequency mapper to stabilize the MIPS rate of OoO processors by DVFS. In Section 6, we show the effects of the controller configuration on the quality of the stabilization. In Section 7, we use simulation to compare the stabilized out-of-order processor with the baseline out-of-order processor. In Section 8, we compare the stabilized OoO processor with an in-order (IO) processor. Sensitivity of the controller design to cache sizes is covered in Section 9. In Sections 10 and 11, we summarize our contributions and conclude.

2. Proposed Framework for Real-Time Applications Development

To apply our proposed framework, the required execution rate (MIPS) of the processor must be known beforehand based on the throughput demands of target applications. This is done by analyzing the code statically and counting the worst-case number of instructions executed for each task. Given the time constraints to complete the tasks, the MIPS rate of the selected processor must be sufficient to meet the deadline of the most demanding task. Then the parameters of the PID controller must be adjusted so that deadlines for all tasks are met right on time, at the lowest possible frequency and without wasting energy in idle cycles. Fortunately, as we will see, the selection of PID parameters is quite robust across hardware platforms.

This process is far easier than a classical WCET analysis. Figure 1 illustrates two different approaches to develop a real-time application. With our stabilization framework shown in Figure 1(b), the development of a real-time application is much simpler than with the traditional approach shown in Figure 1(a) because an accurate mathematical model for the performance of an OoO processor with a cache hierarchy is very hard to develop [15, 34]. OoO processors and hardware controlled caches currently are anathema to real-time application developers. With the stabilization framework, WCET analysis is no longer necessary to guarantee real-time deadlines. Our framework in effect shields the application developer from the intricacies of the microarchitecture. The major issue becomes the choice of a stabilized processor which can maintain the required MIPS rate. In this paper the instruction set architecture is a RISC-like architecture (Simplescalar PISA). Complex ISAs such as the x86 ISA which take multiple RISC-like micro-ops to execute will require more effort since micro-ops execution rate must be stabilized instead of the instruction execution rate. The application developer will have to be aware of the number of micro-ops per instruction to estimate the target micro-op rate. But this would be true whether or not the processor is stabilized and whether the processor is OoO or IO.



(a) Classical approach to develop a RT application

(b) Stabilization framework to develop a RT application

FIGURE 1. Work Flow to Develop Real-Time Applications

3. Related Work

Several research papers have focused on WCET analysis for OoO processors in the hope that they can be deployed in real-time applications [15, 34]. This is a very hard problem because of all the complex and unpredictable features of OoO processors such as speculative execution and hierarchical cache memories. Our approach is not the usual approach of analyzing the contributions of all the architectural features to WCET and of bounding their values in some reliable way, but rather, our

approach is to force the processor to sustain a steady target MIPS rate. If the worst-case number of instructions in the dynamic trace of an application is known, and if we have a stabilized processor with quasi-deterministic MIPS rate, the execution time of the target application can be estimated precisely upfront, by simply counting the worst-case number of instructions needed by the execution of each task.

Power and energy consumption bring up many issues such as hotspots, cooling costs and energy budget, and many studies have addressed them. In [31] the problem of finding the optimal trade-off between heat and performance is resolved by an RC-model based PID controller such that the processor always keeps running within the envelope of the power (heat) budget while pushing performance as much as possible. As the mathematical RC model is well-defined and as the effect of heat dissipation on temperature is a rather slow process, PID control works very well in this context. In [38] feedback control is also applied to reach an optimal real-time schedule of tasks, and to stabilize the processor throughput. The PID controller is configured by trial and error, because the mathematical model of the plant to control is unknown. Our approach is to stabilize the MIPS rate.

In GALS [20] and MCD [10, 29] a micro-architecture is partitioned in several, independent domains, and energy efficiency is achieved by varying the frequency and voltage of each domain inside the processor separately. This approach tries to reach a balance between the activities in every domain of the processor so that the processor can be most power/energy efficient without losing much performance. In [22], the authors target performance throughput for energy efficient, load-balanced architectures in the context of GALS CMPs. Although our approach is limited to the control of the clock and Vdd in the entire processor, it could be extended to architectures partitioned in independent clocking domains, to further optimize the performance/power/energy trade-offs.

The studies in [5] and [16] were starting points for our research. In [16] the high performance variability of OoO processors is explored and one of the conclusions is that IPC is practically independent of processor frequency, a key observation which motivated our interests in adaptive techniques to suppress the high performance variability of modern processors. In [5] energy savings are realized when the processor targets a certain throughput with DVFS. We address the same problem in this paper, but with a different approach to achieve throughput stabilization in the context of real-time systems.

The studies in [6] and [7] showed that the execution time of a real-time high-priority thread in a multi-threaded system can be predictable when it is scheduled with smart resource management. In our research, execution time variability is caused by performance fluctuations due to the complexity of OoO execution. In general, our framework at the hardware, instruction-execution level can be deployed in combination with other software or thread scheduling techniques to improve the overall outcome in various contexts.

Other papers have tackled the same problem as this paper, but with different approaches [17, 22, 27, 36]. In [27] the focus is on improving the execution predictability of an OoO processor by throttling the processor so that it operates at a frequency which avoids violating the deadline. Processor performance is speculated optimistically by slowing down the processor at first and then assessing the slack to the deadline to set the next operating frequency in order to catch up with the remaining work on time. This approach may cause deadline overruns and targets soft real-time tasks. In [17] compiler profiling is first used to schedule tasks optimally. This work is closer to [27] as the framework is based on a measure of slack to the deadline. In [36] PID control manages the DVFS hardware to suppress jitter and improve energy efficiency. This work was made in

the context of a simple IO processor where variability is much less severe than in OoO processors. In OoO processors, we can leverage ILP to compensate for frequency reductions with the goal of improving energy/performance trade-offs.

4. Performance Variability in OoO processors

4.1 Baseline OoO Processor Model

We start with an off-the-shelf OoO processor without DVFS as the baseline. We have modified SimpleScalar/Wattch [3, 4] to model it. We chose a P6-like microarchitecture, which is compatible with SimpleScalar and also is a good candidate for the embedded market. In the power model, Wattch is configured for a 90nm technology. Vdd values for different operating frequencies are adopted from a technical report on Intel's 90nm logic technology used for the Pentium 4 processor [25, 35]. The minimum Vdd is selected for each operating frequency in order to reach the best power efficiency possible. Table 1 gives the major architectural parameters used in the models for the baseline OoO processor. The ISA is SimpleScalar PISA.

TABLE 1. Architectural Parameters for the Baseline OoO Processor Model Clocked at 1GHz

	Parameters
Machine Buffers	40-entry ROB 6-entry IFQ 28-entry LSQ
Machine Width	Maximum 3-wide fetch/decode/issue/commit
Branch Prediction	2-level, local histories, per-set counters 512-entry 128-set 4-way BTB 16-entry RAS at least 5 cycles taken for misprediction
Functional Units (operation latency)	2 INT ALU (1) 1 INT MULT (4) 1 FP ALU (2) 1 FP MULT (5) / DIV (32)
On-chip Memory (line size, latency)	non blocking 16KB 4way IL1 (32B, 2) non blocking 16KB 4way DL1 (32B, 2) non blocking 512KB 8way UL2 (32B, 8) 4-entry fill buffer
Off-chip Memory latency	88
Technology / Vdd	90nm / 0.825V

TABLE 2. Programs Chosen from MiBench

category	**program**	**input**
automotive	basicmath	ref: large
	bitcount	ref: large
	qsort	ref: large
	susan: smoothing	ref: large
network	dijkstra	ref: large
	patricia	ref: large
security	blowfish: decode	ref: large
	blowfish: encode	ref: large
	rijndael: decode	ref: large
	rijndael: encode	ref: large
telecomm	adpcm: adpcm-to-pcm	ref: large
	adpcm: pcm-to-adpcm	ref: large
	crc32	ref: large
	fft: forward	ref: large
	fft: inverse	ref: large
	gsm: toast	ref: large
	gsm: untoast	ref: large

4.2 Benchmarks

To explore the MIPS rate variability of the baseline OoO processor we have run a large number of execution samples representing all possible program phases in the target application domain. We have selected 17 programs in the MiBench benchmark suite. MiBench is a mix of programs for real-time embedded applications. The MiBench suite resembles the EEMBC benchmark suite, which is widely accepted in the commercial embedded system community [13]. The selected 17 programs are a mix of integer and floating-point intensive programs. In order to have execution samples with a rich mix of behaviors, we sliced the dynamic execution of all 17 programs into chunks of 40 millions consecutive retired instructions. These execution samples form the set of tasks to execute within a time deadline in our evaluations. The task size was chosen based on a study [37] showing that the processing of each MPEG-4 frame requires 30 to 42 millions instructions in SimpleScalar. In

total, we evaluate 388 tasks for a total of 1.5 billion retired instructions from the MiBench suite. Table 2 shows the selected 17 programs from the MiBench suite with their inputs set. The MiBench programs are compiled for the PISA ISA and run on Wattch (SimpleScalar/PISA + Power).

4.3 MIPS Rate Variability in the Baseline Processor

Figure 2 shows the dynamic MIPS rates measured from the first retired instruction to the last retired instruction of every task on the baseline processor clocked at 1GHz. The dynamic MIPS rate is computed as the total number of retired instruction (in Millions of instructions) divided by the time (in seconds) since the beginning of the task execution. The dynamic MIPS rate varies widely both across tasks and as a function of time, from 600MIPS to 2.1GIPS.

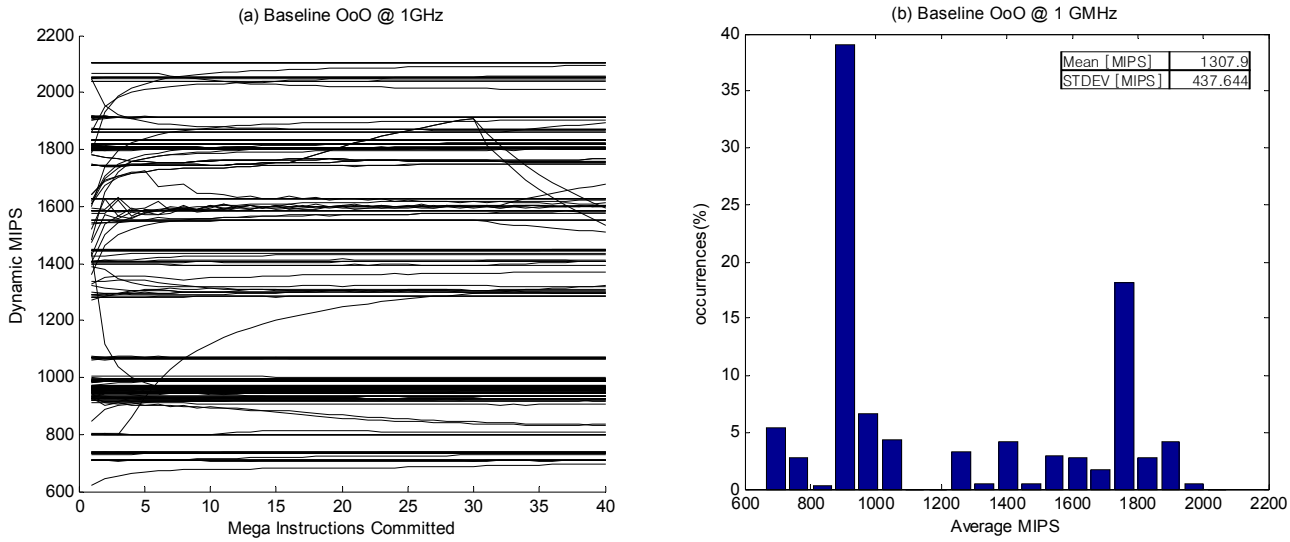


FIGURE 2. Dynamic MIPS Rate (a) and Average MIPS Rate Distribution (b) of All 388 Tasks on the Baseline Processor

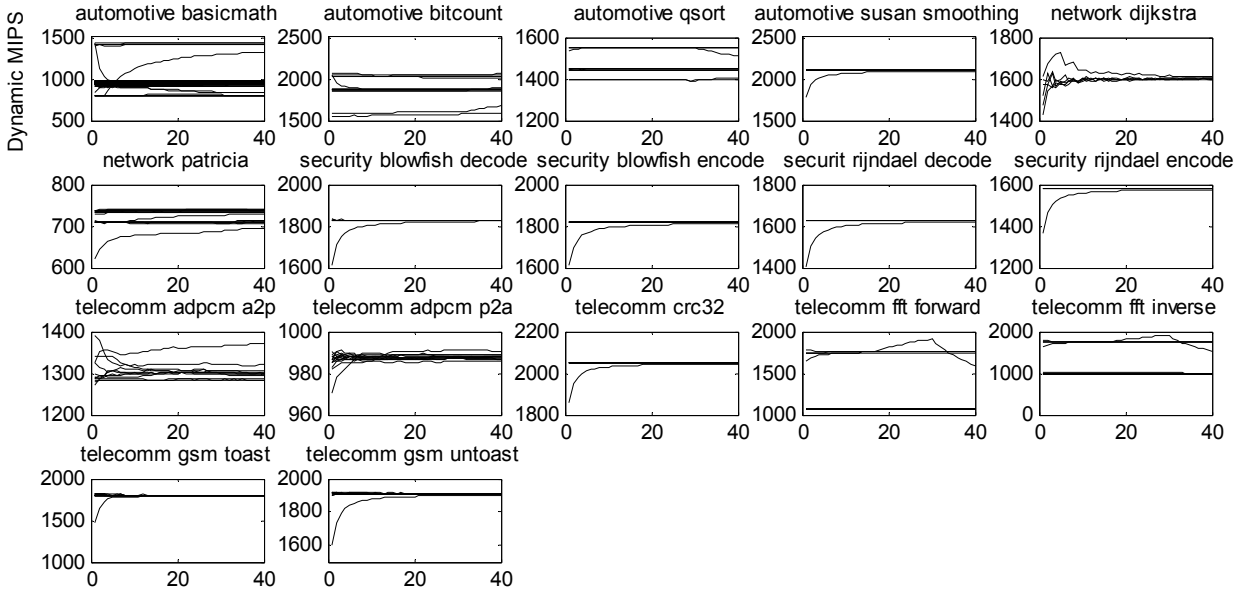


FIGURE 3. Dynamic MIPS Rates of the Tasks of Each Program on the Baseline Processor

Figure 2 also shows the distribution of the average MIPS rates of all 388 tasks run on the baseline processor to the right. The average MIPS rate of each task is computed as 40 divided by the execution time of the task (in seconds). The standard devi-

ation of the MIPS rate (438 MIPS) indicates wide variations around a mean MIPS rate of 1.3 GIPS for the 1GHz baseline processor. Variable memory access latencies, program dependencies, speculative execution and variable ILP all contribute to this high variability. And Figure 3 shows the dynamic MIPS rates gathered for the tasks of each benchmark program on the baseline processor. Among all the programs, *basicmath*, *bitcount*, *qsort* and *fft* have the highest variability across their tasks.

In this paper, we change the on-chip clock rate only. Because off-chip latencies remain unchanged, the L2 miss penalty (in units of processor cycles) is reduced as we slow down the processor clock. Thus lower on-chip frequencies should improve IPC. However, we have observed that L2 misses are rare in our benchmarks, and thus the potential performance advantage from lower speed gap (in cycles) between memory and processor is small. In Section 8 we will explore the effect of cache sizes on stabilization.

In the following section, we describe our approach to stabilizing the performance of the baseline processor.

5. PID Feedback Controller to Stabilize Performance

Among various feedback control schemes, the PID (Proportional-, Integral-, and Differential-gain) feedback controller [12], illustrated in Figure 4 is widely adopted. When the plant, which is the target system controlled by the controller, is mathematically well defined the PID controller can be designed and tuned mathematically with the aid of control theory. However, even when the mathematical model of the plant is unknown, a PID controller can still be designed to work well by fine-tuning the controller parameters empirically. Unfortunately, the characteristic of an OoO processor is neither mathematically well defined nor stable. The plant model changes continuously, depending on the program phase, memory system behavior, execution speculation, and conflicts. In fact, in our framework, the input signal to the system (the reference) is fixed but the plant characteristic keeps changing. This is contrary to the traditional control system design problem.

5.1 PID Feedback Controller

A generic PID feedback control system works like a band-pass filter. One input to the control system is the reference signal. The control system attempts to force the output of the plant (which is the system to be controlled) to track the reference signal. In a feedback control system, the error signal (the difference between output and reference) is relayed by the feedback loop, and is continuously applied to a controller whose function is to bring the error down to zero. If the feedback control system configuration is poor or even wrong, then the information fed back to the controller may drive the system into unstable states. Therefore, careful design of the controller is critical.

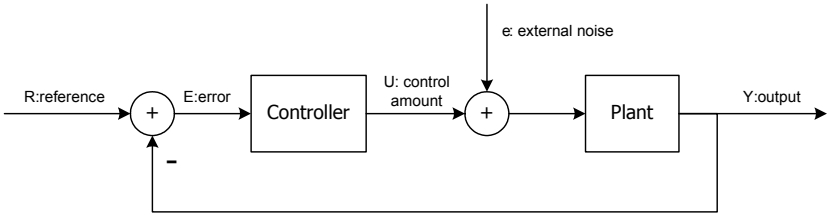


FIGURE 4. Generic Closed-loop Feedback Controlled System

Mathematically, the PID controller is defined by the following equation:

$$U = K_P E + K_I \int E dt + K_D \frac{dE}{dt} \quad \text{(EQ 1)}$$

where K_P, K_I and K_D are the Proportional gain, the Integral gain and the Differential gain, and U is the control input applied to the plant. The PID controller sums up three terms proportional to the value, the integral and the derivative of the error and passes the result to the plant as input. Feedback corrections are continuously applied until the error becomes 0 and thus ideally the output tracks exactly the reference input, with some delay. Equations 2 show the discrete model for the PID controller obtained by a bilinear transformation of Equation 1.

$$U_n = U_{n-1} + \Delta U_n \qquad \text{(EQ 2)}$$

$$\Delta U_n = K_P(E_n - E_{n-1}) + K_I E_n + K_P\{(E_n - E_{n-1}) - (E_{n-1} - E_{n-2})\}$$

5.2 Tuning the PID gains

The values of the three PID gains directly affect the performance of the controller. If the controller reacts too fast to the error it may amplify the error even if the error is very small so that the system tends to move erratically and does not settle down. If the control gains are too large the system may actually become unstable. On the other hand, if the controller reacts too slowly, reaching the target output value by tracking the reference input may take a very long time and convergence may never be achieved. Usually the controller is tuned for the step input. With a step input, the system with an ideal controller is designed to output the same step function as faithfully as possible. By examining the system output under a step input, various observed characteristics of the output such as overshoot, rising time, settling time and steady-state error help tuning the controller properly. The three terms of the PID equation generally have different effects on the output.

- *Proportional term*- By reacting immediately to the current error, this term causes the system to react faster in general. It decreases the rising time and steady-state error. But it also causes larger overshoots.
- *Integral term*- By reacting to the integration of the error from past to present, this term lags behind the system phase in general, thus increasing the settling time. It makes the system insensitive to sudden changes. This term is critical to eliminating the steady-state error.
- *Differential term*- By reacting to the derivative of the error, this term leads the system phase in general. Thus it decreases the overshoot or the undershoot and makes the system settle faster. However it makes the system sensitive to small variations, even to noise.

When the plant characteristic cannot be specified mathematically, a trial-and-error approach called loop-tuning is used to tune each gain.

5.3 Throughput-to-Frequency Mapper

The description of the PID controller above assumes that all the control variables reside in the same domain. In our context (the stabilization of processor throughput), the variable to control is the MIPS rate but the variable effecting the control is the frequency (MHz). In order to convert MIPS error into frequency change, a throughput-to-frequency mapper is included in the feedback loop. Figure 5 shows the system with the mapper added. Note that there is no external noise and the resynchronization penalty is associated with plant when the frequency changes.

To compute the next frequency the mapper assumes that the processor has the same behavior in the next phase as in the current phase and that the IPC does not change with the frequency. It computes the next on-chip operating frequency as:

$$Next\_Freq = Current\_Freq \times Target\_MIPS \,/\, Current\_MIPS \qquad \text{(EQ 3)}$$

In a DVFS scheme, the number of possible frequencies is limited. Thus the mapper must map the next operating frequency to one of the operating frequencies provided by the DVFS scheme. Figure 6 shows the mapping of target (continuous) frequencies to discrete frequencies in the DVFS scheme. Each discrete frequency in the DVFS scheme is assigned to a range of target frequencies. The frequency assigned by the DVFS scheme is not necessarily at the center of its associated range of target frequencies. By slanting the DVFS frequency towards the higher end of its associated frequency range (as shown in Figure 6), a given target frequency tends to map to a higher DVFS frequency. Thus the system response is stronger when the MIPS rate falls below its target, and it is weaker when the MIPS rate is above its target.

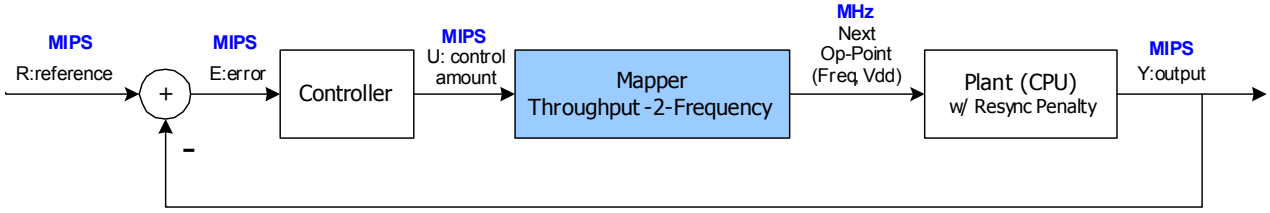


FIGURE 5. Controller with Throughput-to-Frequency Mapper

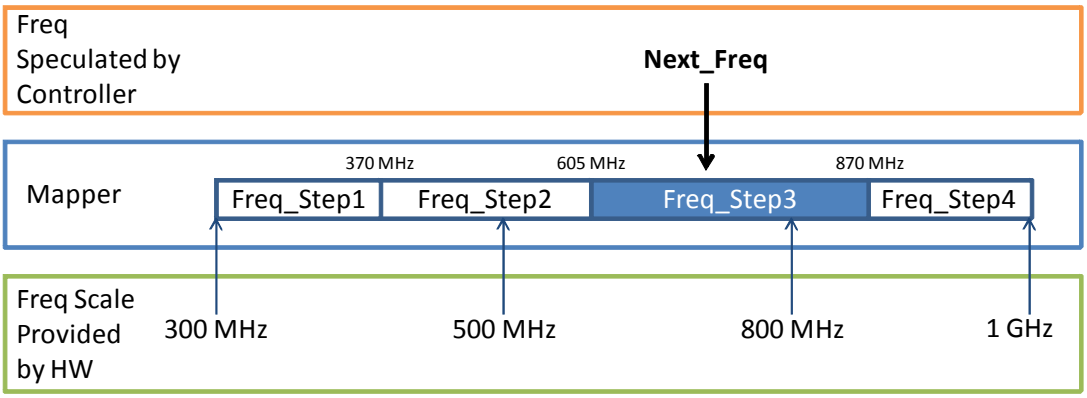


FIGURE 6. Configuration of the Discrete Frequency Mapper

6. Controller Design

In this section, we show how we arrived at the final controller design using a reduced set of 11 tasks called the *training* set. The reason for the training set is that exploring the design space over all 388 tasks is not feasible. The first step is to explore the effects of the controller parameters (gains) on the MIPS rate stabilization, given a DVFS framework.

6.1 DVFS Framework

The four voltage and frequency value pairs in our DVFS framework are shown in Table 3. These frequency and voltage pairs are drawn from Intel data [25, 35]. On each voltage/frequency transition the processor pauses for 20 microseconds due to PLL resynchronization [10]. Voltage switching is much slower than frequency switching. In [10] DVFS transition speed was measured to be 10mV/microsecond. Our DVFS configurations swing within a 200mV range, hence the 20microseconds pause. Furthermore, we assume that the energy consumption due to each voltage/frequency transition is negligible based on observations made in [30, 33]. Similar assumptions were made in many DVFS studies [5, 10, 11, 17, 20, 22, 24, 26, 27, 36, 38].

A task execution is divided into non-overlapping *control windows*. During each control window, the controller collects information on the current throughput and determines the operating point for the next control window. In this study the control

window is set to 50K retired instructions. Whenever the control window ends, after 50K instructions have committed, the error communicated to the controller is the difference between the reference throughput and the average instruction throughput in the window. The controller then determines the next target throughput, and the throughput-to-frequency mapper maps it to the next operating frequency and voltage.

The size of the control window is a design trade-off. When it is small (such as our window of 50K instructions) the controller reacts faster. The potential performance cost is more resynchronization pauses. In this research, there are 800 control windows in each task of 40M retired instructions. Figure 14 displays the average number of resynchronization in all the tasks of each benchmark for the PID controller design we finally selected. Our final controller design causes an average of 120 resynchronizations per task, i.e. an average of 3 microseconds of resynchronization stall per control window. Each control window in the baseline at maximum frequency with no stabilization takes an average of 33 microseconds. Thus the total resynchronization overhead is at most 10% of the execution time of the baseline running at maximum frequency. It is much less when stabilization is deployed given the lower average frequency.

TABLE 3. DVFS Frequency and Voltage Pairs

Frequency (MHz)	Vdd (V)
1000	0.825
800	0.772
500	0.694
300	0.641

TABLE 4. Six Different Parameter Settings for Controller Gains

	K_P	K_I	K_D
Setting 1: Slowest	1	10	0.1
Setting 2	10	50	0.1
Setting 3	50	50	0.1
Setting 4	75	50	0.1
Setting 5	100	50	0.1
Setting 6: Fastest	100	100	0.1

The most important problem of PLL resynchronization is that the processor is stalled during the resynchronization and cannot react to interrupts. This can cause a real-time interrupt loss if the interrupt arrives during the resynchronization period. In real-time systems with hard deadlines, Intel's aggressive XScale-style resynchronization can avoid this problem, since it does not stall the circuit when the frequency and voltage change [8, 28, 29].

6.2 Effects of Control Parameters on Stabilization

If the controller responds faster and often changes the frequency, the processor throughput is more stable. However every time the frequency and voltage change, the processor is stalled during PLL resynchronization. The combined performance penalties of PLL resynchronizations lower the maximum stable MIPS rate attainable by the stabilized processor and thus narrow its target MIPS range. Furthermore the faster the controller is, the higher the power because fast controllers switch to higher frequencies more often than needed.

On the other hand, if the controller is too slow, the system deviates from the target throughput and takes a long time to converge, albeit at a lower power cost. Energy consumption is related to both power consumption and execution time. Execution time is inversely proportional to throughput, whereas dynamic power decreases superlinearly with frequency. Thus we expect that lowering the target throughput will result in lower EPI (energy consumption per committed instruction).

The parameter settings we have explored for the controller are shown in Table 4. Different settings affect the performance of the stabilization. As the mathematical model of the plant is unknown, it is safer to keep K_D small in order to prevent unexpected divergence as the differential gain parameter detects and amplifies any fast phase jitter.

Figure 6 shows the configuration of the discrete frequency mapper used in this study. The mapper is biased towards higher frequencies to provide some performance margin especially when the characteristic of the plant changes abruptly.

We have modified SimpleScalar/Wattch to include the DVFS scheme, the throughput-to-frequency mapper and the PID controller. The stabilized processor is exactly the same as the baseline OoO processor, except that it is equipped with DVFS and PID control. We have simulated the six operating points shown in Table 4 for target MIPS rates of 200MIPS to 1400MIPS.

To reduce the complexity of the space search for the controller design, we first picked the most representative 11 tasks using SimPoint [14] in the four programs - *automotive_bitcount*, *automotive_qsort*, *telecomm_fft_forward* and *telecomm_fft_inverse* - with the highest performance variability among the MiBench programs of Table 2. This training set of 11 samples has high throughput variability. Thus if the controller can stabilize this reduced set of tasks, we surmise that it can also stabilize the whole set of 388 tasks.

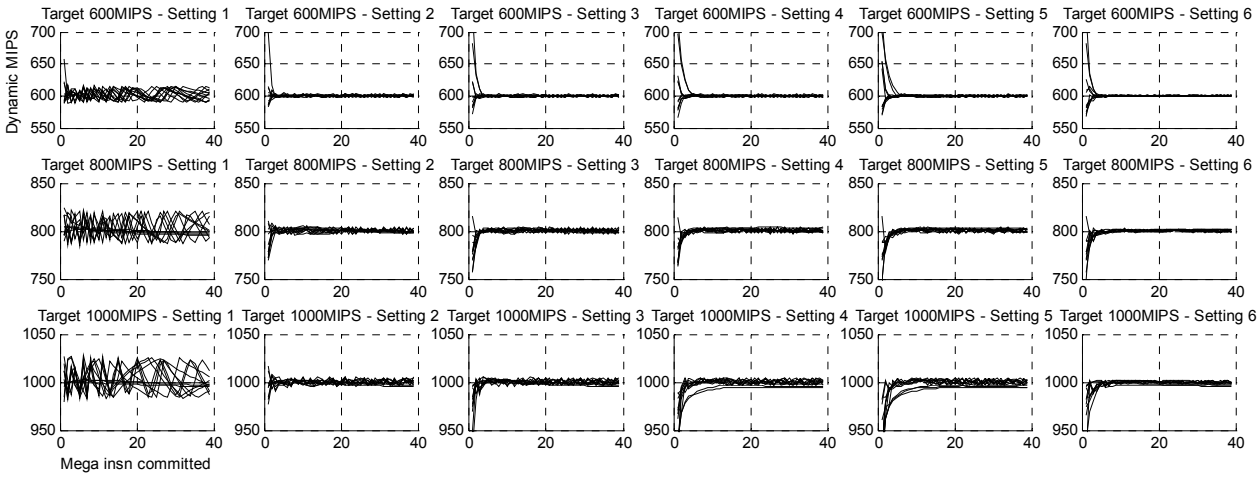


FIGURE 7. Dynamic MIPS Rate Stabilization for Training Set - Targets of 600, 800 and 1000 MIPS

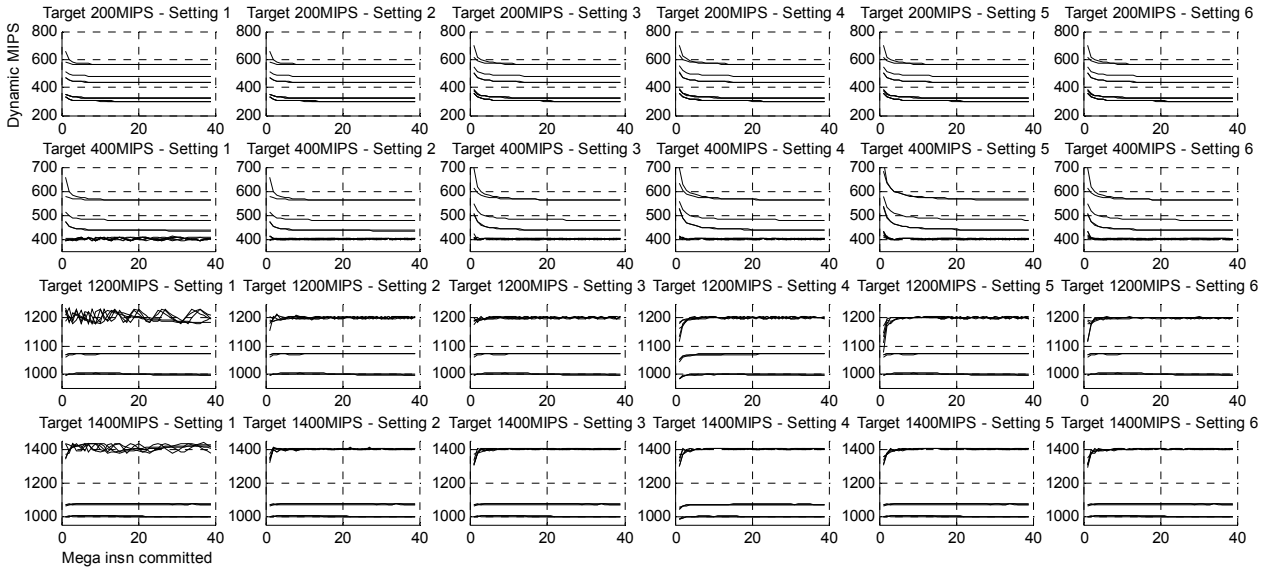


FIGURE 8. Dynamic MIPS Rate Stabilization for Training Set - Targets of 200, 400, 1200 and 1400 MIPS

Figure 7 shows the dynamic MIPS rate of the stabilized processor for the training set of 11 tasks and for the six controller settings with target MIPS rates of 600, 800 and 1000MIPS. We observe that the throughput of the stabilized processor con-

verges rapidly to its target MIPS rate, except for the slowest controller with parameter settings 1. It appears that the controllers work slightly better for a target throughput of 600MIPS, but they all meet the goal of stabilizing the MIPS rate for targets of 800 or 1000MIPS.

We further ran the stabilized processor with the different parameter settings and MIPS targets of 200, 400, 1200 and 1400MIPS. The results for the training set of tasks are shown in Figure 8. We observe that in all cases the throughput of several tasks do not settle at the target MIPS rate, for various reasons. When targeting 200MIPS, the stabilized processor keeps running at the minimum frequency of 300MHz all the time, and, because of ILP, the processor exceeds the 200MIPS target for all tasks. When targeting 1200 or 1400 MIPS, some tasks with very low ILP cannot reach the target MIPS rate although the processor continuously runs at its maximum frequency of 1GHz. In hindsight, this observation could have been predicted from a careful interpretation of Figures 2 and 3. Clearly, the stabilized processor designs adopted in this paper cannot reliably stabilize processor throughput at 200, 400, 1200 or 1400 MIPS.

It is remarkable that the behaviors of the controllers are uniform for a wide range of parameter values, across a wide range of target MIPS rates and a wide range of tasks. This is encouraging, as it shows that the design of the controller is very robust and the same design can be re-used in different environments without tedious empirical tuning. We will see more evidence of this controller design robustness when we evaluate the sensitivity of the stabilization to cache sizes in Section 8.

6.3 Effects of Control Parameters on Power and EPI

The major advantage of processor throughput stabilization (besides execution time predictability in real-time applications) is the reduction in power and energy consumed per instruction (EPI).

Figures 9(a) and (b) show the average dissipated power and Figures 9(c) and (d) show the energy consumption per retired instruction (EPI) as a function of the target MIPS rate and for every controller setting 1-6. All values are normalized to the baseline system. We measure energy consumption up until a task is finished, i.e. when 40M instructions have committed. Thus the energy spent by the processor in the idle state, waiting for the deadline to expire, is not included in the EPI. Power and EPI are averaged over all 11 tasks in the training set.

The dotted-line boxes include the design regions where the throughput does not reliably converge to the target MIPS rate because the target throughput was set too low or too high as observed in Figure 8. The solid-line boxes include the regions where the throughput is successfully and reliably stabilized to the target MIPS rate as observed in Figure 7.

Faster controllers (closer to parameter setting 6) trigger changes of operating frequency and voltage more often, and need to offset this performance penalty with higher frequencies. For this reason faster controllers tend to consume more power than slower controllers. Faster controllers also consume more energy per instruction.

However, overall, controller parameter settings do not affect the power or energy of the stabilized processor significantly. Rather, process technology and the DVFS scales are more relevant to power and energy. In Figure 9, we observe that the values taken by power and EPI across all parameter settings do not deviate by more than 6.7%, and 6.9%, respectively (much less still 4% and 3.5% if we discard parameter setting 1 as unacceptable and focus on settings 2-6.) This conclusion again shows the robustness of the controller design in the sense that there is little incentive to optimize the control parameters for power or EPI within a wide range of parameter values and target MIPS rates.

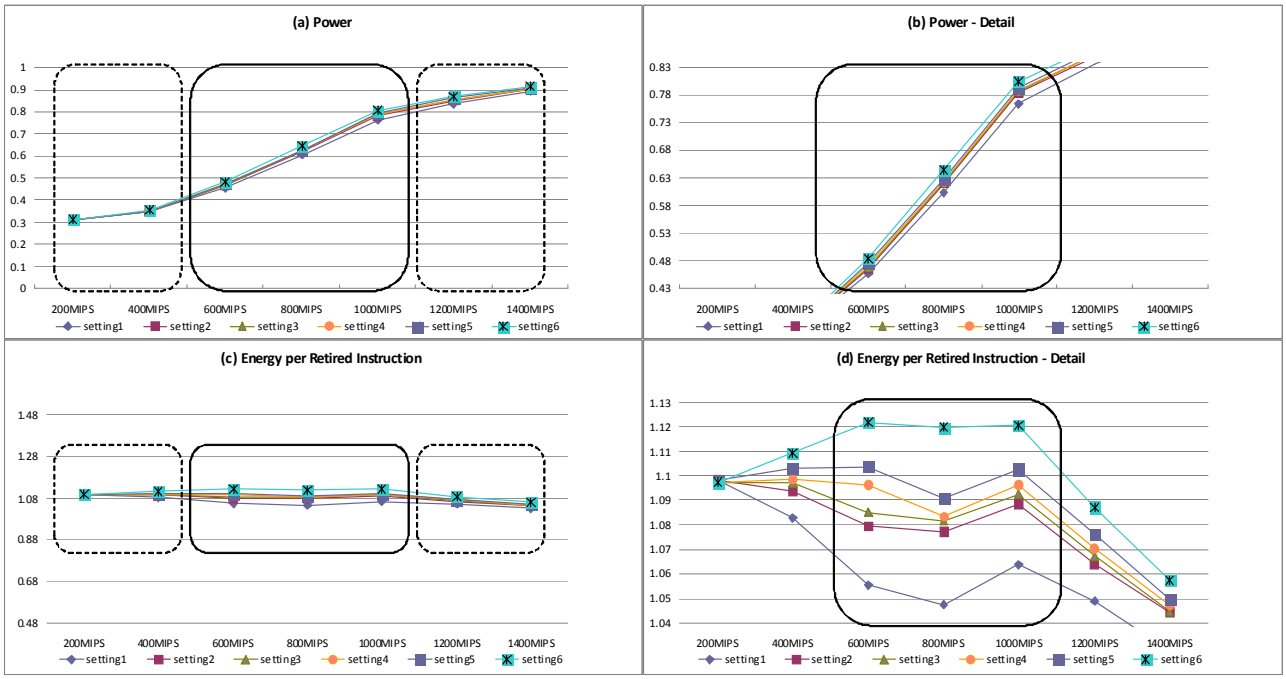


FIGURE 9. Power and Energy per Instructions (training set)

7. Stabilization Results for All Tasks

Based on the previous design and evaluations, it appears that parameter settings from setting 2 to setting 6 are equally acceptable for the training set of 11 tasks. In this section, we show detailed simulation results when the operating point of the controller is set to (K_P, K_I, K_D) = (75, 50, 0.1), i.e., setting 4 in Table 4. We now evaluate this PID controller for ALL 388 tasks which form our workload. The target MIPS is 650MIPS.

The effectiveness of this PID controller is demonstrated by comparing Figures 10 and 11 (processor with stabilization) with Figures 2 and 3 (no stabilization). Figures 10 and 11 show the dynamic MIPS rates of the baseline OoO processor stabilized to the 650MIPS target for all 388 tasks. The MIPS rates of most tasks settle down at the target throughput of 650 MIPS within the first 5 millions committed instructions. Out of the 388 tasks, three chunks do not converge to 650MIPS and finish early. These three chunks are part of *automotive_bitcount*, *telecomm_fft_forward* and *telecomm_fft_inverse* and have high IPC. They exceed the 650MIPS target even if they run at the lowest frequency (300MHz). The deadline is still met in these three cases.

Figure 10 also displays the distribution of the average MIPS rate of all 388 tasks measured on the stabilized processor to the right. It shows that practically all the stabilized MIPS rates are within ±3 MIPS of the target throughput of 650MIPS. The execution statistics of the stabilized processor are also listed. Performance variability has been decreased dramatically (as compared to Figure 2) because of the stabilization. Moreover power consumption is improved. The stabilized processor consumes 46.6% of the power needed by the baseline processor. Since the average MIPS rate of the stabilized processor is 49.7% of the average MIPS rate of the baseline processor, the overall power savings is higher than the overall performance loss.

The number of cache misses per committed instruction and the IPC of the stabilized processor are almost identical in the baseline and in the stabilized processors (not shown in this paper). As previously discussed, the number of cache misses is

very low and the possible IPC gains due to smaller L2 miss penalties (in cycles) when the processor throttles down to lower operating frequencies are less than 1%. As pointed out in [5], the IPC of an OoO processor is practically independent of its operating frequency.

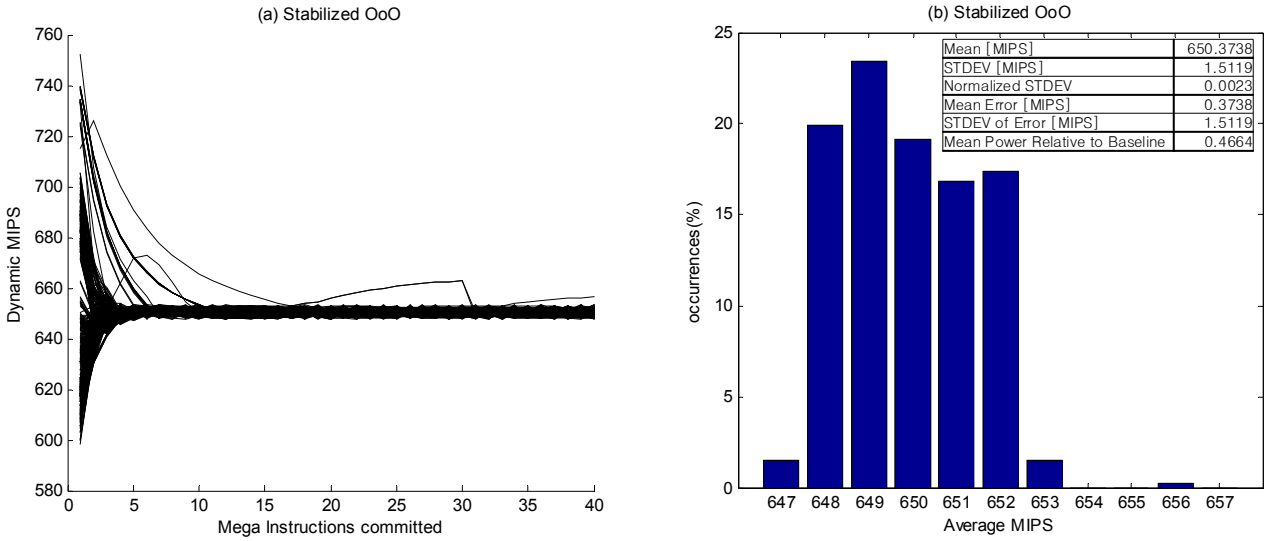


FIGURE 10. Dynamic MIPS Rate (a) and Average MIPS Rate (b) Distribution of All 388 Tasks on the Stabilized Processor

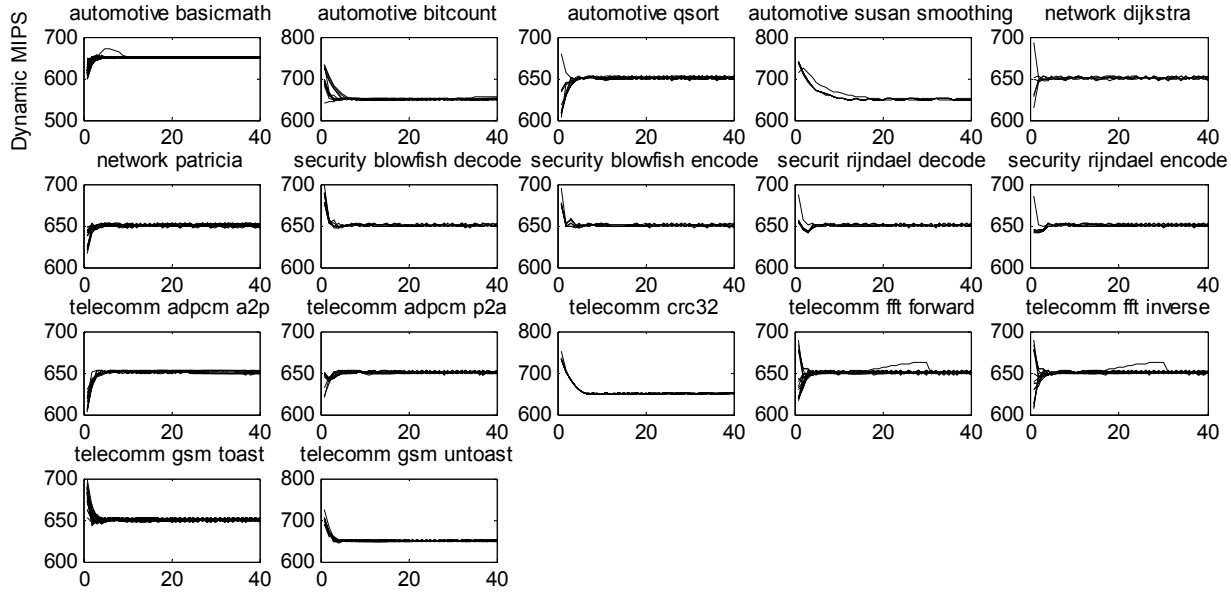


FIGURE 11. Dynamic MIPS Rates of the Tasks of Each Program on the Stabilized Processor

Figure 12 shows the fraction of tasks whose dynamic MIPS rate falls within a margin of the target MIPS rate (i.e., 650MIPS) as a function of the time expressed in number of retired instructions. After the first 5 millions retired instructions, the MIPS rates of practically all the chunks are higher than 647.4MIPS, i.e. within a -0.4% margin of the target. No chunk ever goes below 646.75MIPS, i.e., within the -0.5% margin of the target after the first 5 millions of retired instructions. Therefore, with this controller setup, it is recommended to have at least a 0.3% marginal offset for the target MIPS rate in order to converge to the target MIPS rate as quickly as possible. For example one should use a target MIPS of 652.6 instead of 650 to converge to the target MIPS rate before the first 6 millions retired instructions.

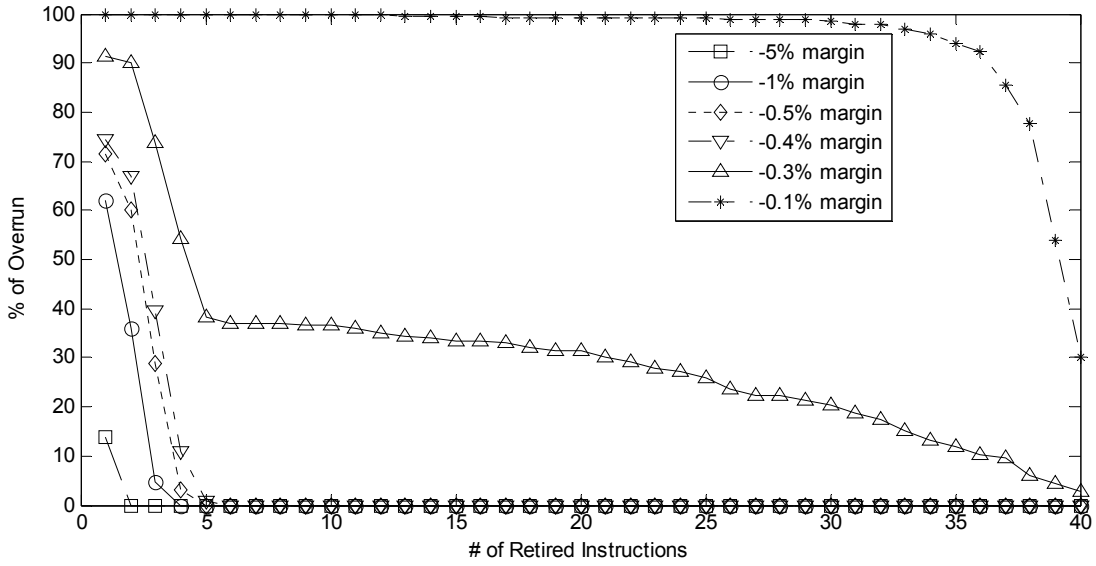


FIGURE 12. Fraction of Tasks Falling Within a Margin of the Target as a Function of Time

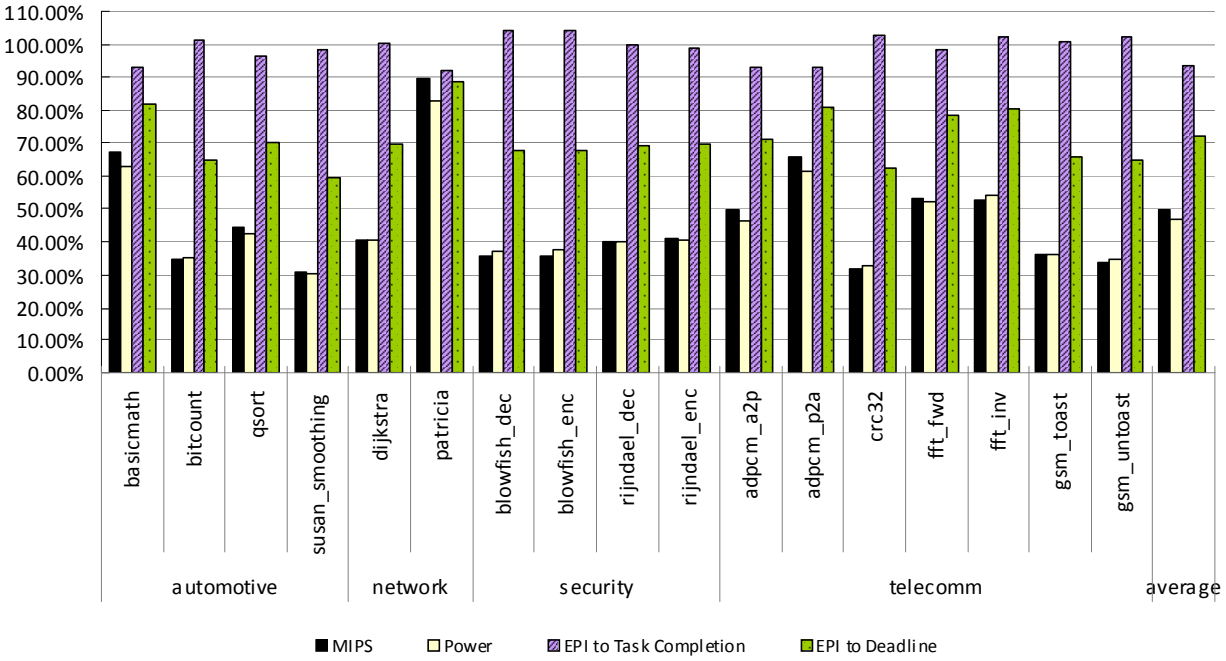


FIGURE 13. MIPS rate, Power and EPIs of Stabilized Processor Relative to Baseline Processor

Figure 13 shows (from left to right) the MIPS rate, Power, EPI (energy per instruction) to task completion and EPI to task deadline of the stabilized processor relative to the baseline processor. Each bar is the average over all tasks of each benchmark program. The baseline processor completes each task early in most cases, while the stabilized processor completes its tasks right on time to meet the deadline. One major contribution to EPI in the baseline processor is the power consumed after the completion of a task and until its deadline expires. During this idle time, static and dynamic power is consumed. In [9] it is shown that the idle power consumption of the XScale PXA255 micro controller is about 29.6% of the average CPU consumption. From Figure 13, we observe that, on the average, the average MIPS rate of the stabilized processor is 49.72% of the baseline, while its average power and EPI to deadline are 46.64%, and 72.06% of the baseline respectively. Note that waking up from the SLEEP state in order to suppress the idle power requires long transition time (160ms for StrongARM

SA-1100). Thus use of SLEEP state might not be viable to high performance real-time applications [2].

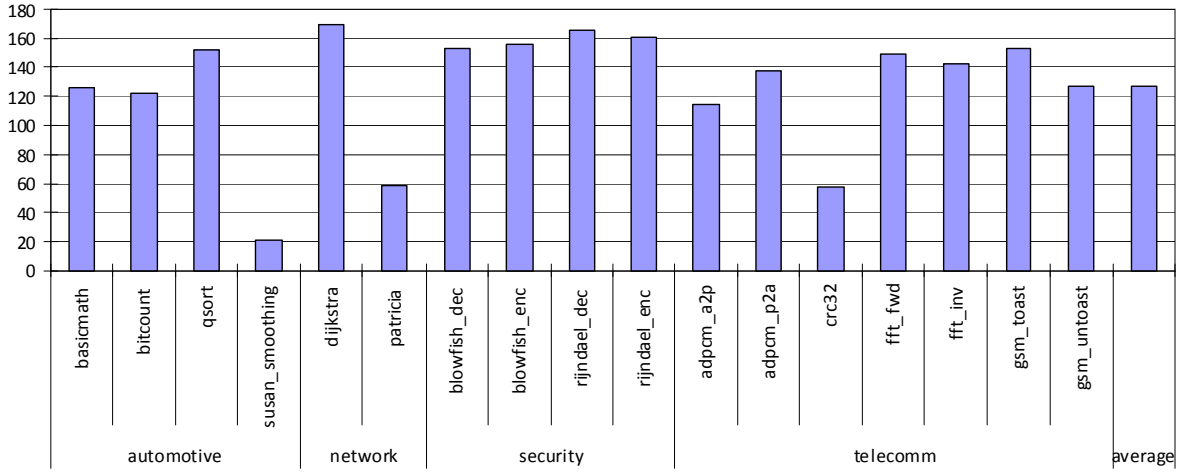


FIGURE 14. Number of Resynchronizations in the Stabilized Processor (Average per Task)

The throughput and power numbers are directly correlated, as expected. Without counting the power dissipated in the idle state we see that very little energy is saved per instruction by stabilizing the processor. This is counter-intuitive, since throughput is proportional to the frequency while (dynamic) power is inversely proportional to the cube of the frequency in our schemes. The low average gains on EPI to task completion are due to two main factors: 1) when the target MIPS rate is much lower than the baseline MIPS rate of the task, the power savings is not really cubic with the frequency scaling factor because static power dominates and 2) the throughput penalty caused by resynchronizations not only wastes time but also pushes up the frequency higher than otherwise to meet the deadline.

If the target MIPS rate is lower than the baseline MIPS rate, the energy efficiency of the stabilized processor is worse than the efficiency of the baseline. Programs with high ILP such as *automotive_susan_smoothing* and *telecomm_crc32* do not fare well with respect to to EPI. Their baseline MIPS rates are high because of their high ILP, and the target of 650MIPS imposed by the controller forces them to run at the lowest operating frequency for long periods of time. This happens because the thermal (power) optimal point and the energy optimal point are different as discussed in [23, 39]. In contrast, programs with low ILP such as *automotive_basicmath* and *network_patricia* take advantage of the stabilization to reduce their energy consumption as the targeted MIPS rate of 650MIPS is close to their baseline MIPS rate.

When stabilization causes frequent frequency changes, more time is wasted during resynchronizations and this effectively degrades the MIPS rate while setting the average operating frequency higher than necessary. The average number of resynchronization per task in each benchmark is shown in Figure 14. The average number of resynchronizations per task is low considering the size of each task (because the frequency rarely changes after each control window of 50K instructions. However some benchmarks are affected more than others. For example, in benchmarks such as *network_dijkstra* and *security_blowfish*, the EPI is worse than in the baseline in part because the stabilization triggers too many frequency changes.

The observations above imply that savings in EPI to task completion can be achieved while stabilizing the MIPS rate provided the target MIPS rate is “close” to the MIPS rate of the baseline. Of course the baseline MIPS rate must be at least the target MIPS rate. If the baseline MIPS rate is highly variable and/or significantly higher than the target MIPS rate, then more

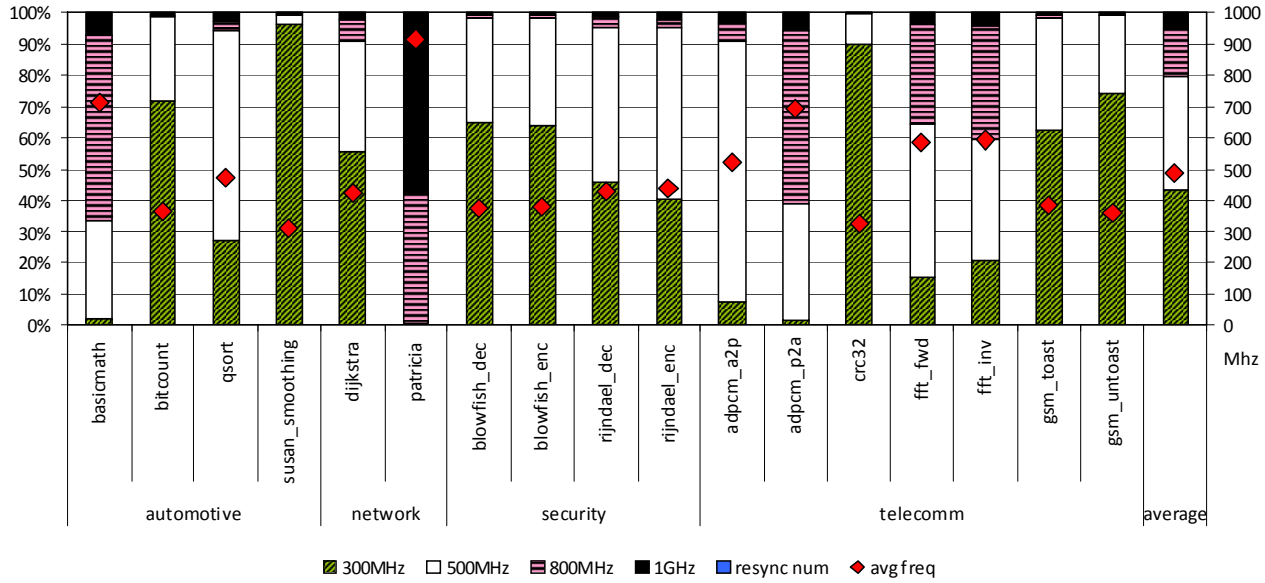


FIGURE 15. Operating Frequency Distribution in the Stabilized Processor

energy per instruction (to task completion) may have to be spent to stabilize the processor.

When power/energy consumption is highly critical in a design, slower controllers may be the better solutions. For example guaranteeing that the system works within the -0.3% margin all of the time requires much faster controllers, which negatively impact the power/energy/performance trade-off as shown by the shape of the curves in Figure 12. A better way is to increase the target MIPS rate by 0.3%. The slower response time of the controller can be easily compensated by raising the target MIPS rate slightly, thus improving the energy/power efficiency at the same time.

Figure 15 shows the average fraction of time spent at each clock frequency in the stabilized processor. *automotive_qsort* and *telecome_crc32* which have high ILP remain at the lowest frequency for most of the time while *network_patricia* which has the lowest ILP remains at the highest frequency for more than a half of the execution time. The average operating frequency of the stabilized processor across all tasks is 485Mhz.

8. Comparison with a Single Issue In-Order Processor

Single issue in-order processors are currently preferred for real-time/embedded applications, because they consume less power and are more predictable for WCET models. However, the advent of media-based embedded systems calls for processors with higher computing power. In Figure 15, it was shown that the stabilized OoO processor running at an average 485MHz can meet the target 650 MIPS rate reliably within a few millions instructions and thus can meet hard deadlines requiring that level of throughput for tasks of more than a few millions instructions. To achieve the same real-time performance level, a StrongARM SA1110-like single issue in-order processor with the same cache hierarchy as our baseline processor would have to run continuously at 1.4GHz.

To reach this conclusion, we selected 4 benchmarks (*automotive_basicmath*, *network_patricia*, *telecomm_adpcm_p2a* and *telecomm_adpcm_a2p*) from Table 2 with the lowest IPC. Figure 16 shows the MIPS rates of a non-stabilized IO processor capable of maintaining a minimum throughput of 650MIPS on all tasks in the four benchmarks. With the same 90nm technology, the average power/EPI to task completion of the IO processor are 151.46%/110.49% of those of the stabilized pro-

cessor, and 86.19%/102.64% of those of the baseline processor.

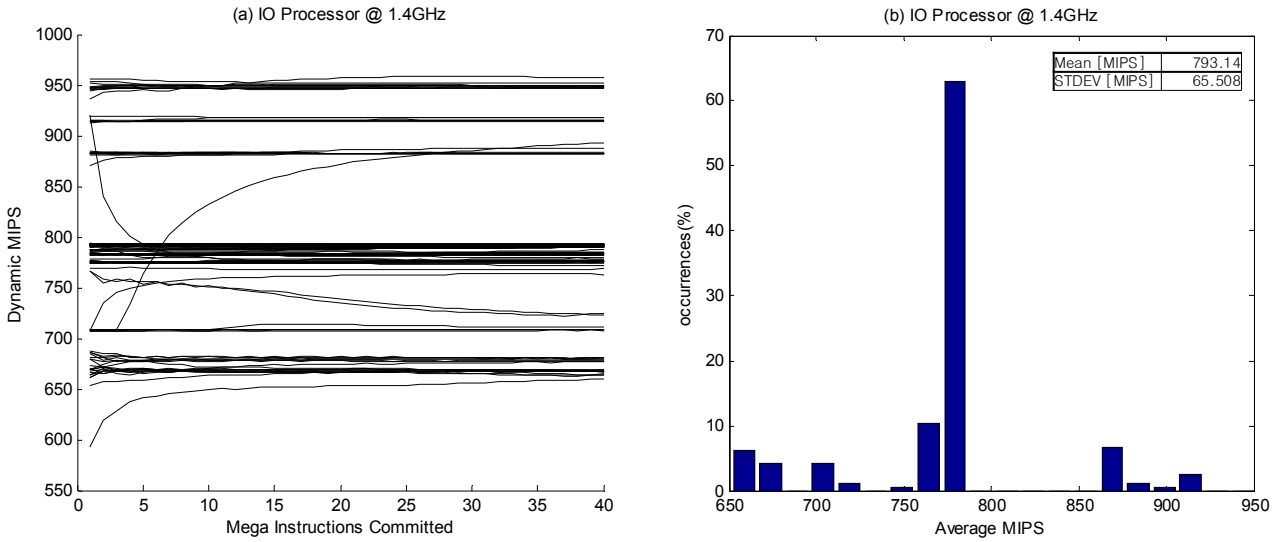


FIGURE 16. Dynamic MIPS Rate (a) and Average MIPS Rate (b)Distribution for the IO processor running 1.4GHz

9. Sensitivity to Cache Sizes

Cache effects are a major cause of execution time variability in real-time systems. Usually caches are software controlled or even disabled to simplify WCET analysis. Cache hierarchies affect IPC and MIPS rate and their effects can be stabilized with the same framework developed in this paper.

TABLE 5. IPC and Cache Misses Per Kilo Instructions in 3 Different Hierarchical Caches

		basicmath_qsort	network_patricia	average
IPC	16K-16K-512K	1.4647	0.7247	1.0947
	8K-8K-256K	0.9802	0.6436	0.8119
	2K-2K-64K	0.8697	0.4911	0.6804
IL1 MPkI	16K-16K-512K	0.8022	88.6539	44.7280
	8K-8K-256K	60.0970	136.8662	98.4816
	2K-2K-64K	79.5046	198.8028	139.1537
DL1 MPkI	16K-16K-512K	2.1870	0.5827	1.3848
	8K-8K-256K	2.9269	5.9971	4.4620
	2K-2K-64K	3.6642	9.9585	6.8113
L2 MPkI	16K-16K-512K	0.8710	0.1401	0.5056
	8K-8K-256K	1.3983	0.1818	0.7900
	2K-2K-64K	1.6632	4.4996	3.0814

To evaluate the sensitivity of the controller design to cache sizes we evaluate two systems with reduced cache sizes. We run simulations of our stabilized OoO processor with caches reduced by a factor two and four. We select 2 benchmarks (*basicmath_qsort* and *network_patricia*) from Table 2 that show highest cache misses per instructions. The average IPC and cache miss rates for all tasks in the two benchmarks are shown in Table 5. IL1, DL1 and L2 misses per kilo-instructions increase 3, 5 and 6 times on the average when the cache sizes are reduced by a factor four. Furthermore the IPC of the baseline processor is reduuced to 62.154% of the original stabilized processor. Because of these lower IPCs, the target MIPS rate must be adjusted, so that the slowest tasks can meet their deadlines. The target MIPS rates are set to 550 and 450 for the systems with cache sizes reduced by 50% and 75%. We keep the same parameters for the controllers.

Figure 17 shows the dynamic MIPS rates of tasks gathered from those benchmarks and the distribution of the average MIPS rates of all tasks. With the same controller parameters, the variance of the average MIPS rates of the stabilized processor is similar to the variance obtained for the stabilized processor with original caches sizes. The MIPS rate still converges quickly to its target, within 5 million instructions of each task. This is further evidence that the design of the controller is robust and valid across different cache sizes.

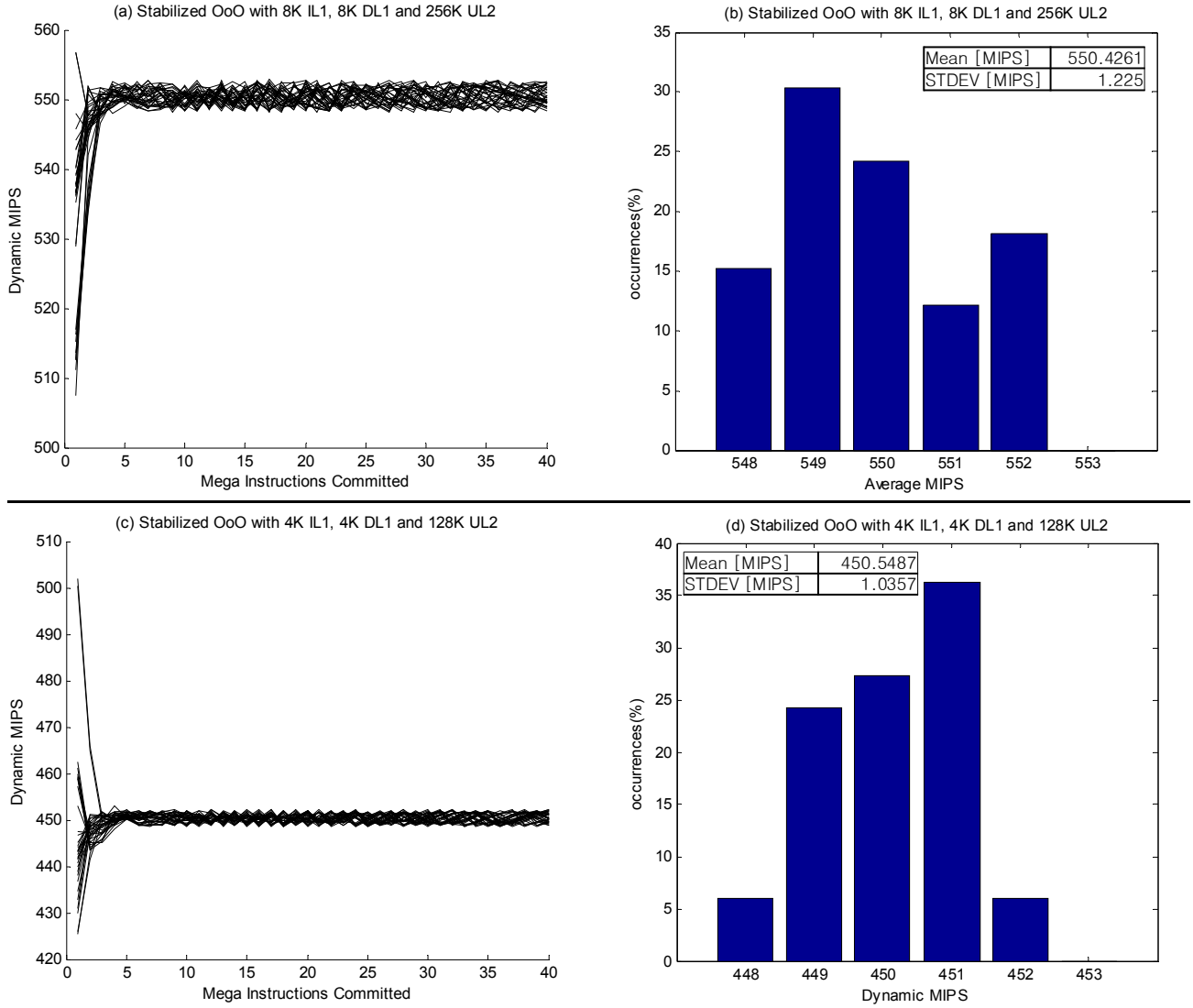


FIGURE 17. Dynamic MIPS (a), (c) and Average MIPS Rate (b), (d) Distribution with Smaller Caches

10. Discussion

We have shown that stable and robust controllers can be designed to maintain a predictable MIPS rate for tasks of more than a few million instructions. Whereas the reliability of the stabilization cannot be proven mathematically as the WCET analysis possibly could, there are several reasons why our stabilization framework is preferrable to WCET analysis.

WCET analysis has not been successful in the context of OoO micro-architecture and hardware-controlled caches because mathematically modelling of the complexity of OoO processors with hierarchical caches is very difficult. Actually this complexity is the reason why the mathematical model of the plant in our control system is intractable and we cannot come up with a design based on control theory. Applying the stabilization framework is, however, very easy and can be done by ana-

lyzing the target tasks, as demonstrated in this paper. Tuning the control parameters empirically can be time-consuming but it is also one part of the normal development process, as real-time embedded systems are heavily optimized in general. By selecting a small training set of tasks the complexity of the design space exploration is significantly simplified. The controller design is very robust and does not require fine tuning.

WCET analysis is also expensive and any small changes in the target processor may cause a rework of the WCET model. Furthermore, if the processor has adaptive features such as DVFS to solve thermal, power or energy issues, the correct WCET model is even harder to derive. In future micro-processors, we expect that such adaptive mechanisms will become pervasive. In our stabilization framework, such unpredictable execution rate variations are handled automatically, provided the processor can meet the deadline of the slowest task. Target and processor must be selected carefully. To apply our hardware-based stabilization framework the target tasks must be profiled, but this procedure can be easily automated.

We targeted 650MIPS in this paper because we wanted to demonstrate that a single controller design could be deployed for a wide set of tasks. In practice, it is common that the system runs only one or a couple target programs repetitively. In that case, the target MIPS rate can be higher. For example, if the system would only run *telecomm_adpcm_p2a* or *telecomm_adpcm_a2p* (see Figure 3) then the stabilization framework could target 1280MIPS or 985MIPS respectively. Note that the framework is flexible enough to let the thread (or process) scheduler arbitrarily change the target MIPS rate on the fly. All in all, the successful application of the stabilization framework rests on careful profiling of the target applications.

11. Conclusion and Future Work

We have proposed to stabilize the throughput of out-of-order processors with hardware-controlled cache hierarchies to improve performance predictability, power dissipation and energy efficiency. The framework to stabilize throughput is a simple PID feedback controller with DVFS (dynamic voltage frequency scaling technology). A simple DVFS scheme with only four steps is sufficient for a wide range of stabilization target throughputs. The simple PID-control based stabilization framework we have proposed and demonstrated in this paper can be implemented by software in the OS layer or by hardware with simple discrete time controllers widely used and available in the control market.

We have shown that the processor throughput can be stabilized within a very narrow band near the target throughput in less than a few million retired instructions. This enhances the performance predictability of the processor, and reduces heat dissipation by lowering the power consumption. If the target MIPS rate is well-chosen, the EPI to task completion can be reduced as well. If the program execution rate varies moderately in the baseline, then the stabilization degrades performance very little while yielding power and energy savings on top of the improved performance predictability. Furthermore, with stabilization, the overall average EPI is 72% of the baseline if we count the energy wasted in idle mode in the baseline while waiting for the expiration of the deadline.

We also showed that faster controllers can be detrimental to the performance/power/energy trade-off. Rather it is better to raise the target MIPS rate by small margins and slow down the controller.

There are several research avenues to improve the results presented in this paper. Better hardware support such as adaptive feedback controller, and finer-grain DVFS steps with Intel’s XScale frequency and voltage resynchronization technology

would most probably improve the stabilization with minimal penalty. The target MIPS rate could be changed dynamically whenever the application needs higher throughput, although, in this paper, we only explored systems with constant target MIPS rates. PID parameters could be tuned adaptively on-the-fly as well, depending on how critical the settling time is. We think that this framework can be adopted in GALS systems, MCD architectures or CMPs to manage individual components or domains effectively.

By its nature, the downside of the stabilization is that the baseline processor is always faster, as it can freely take advantage of all ILP opportunities with the highest clock rate possible. However, if ever the maximum performance of the baseline processor is necessary, stabilization can be easily turned off.

With our stabilization framework, it is possible to hide the details of modern microarchitectures for real-time application developers. Developers need to select a processor which can maintain the required MIPS rate and determine its stabilization parameters instead of relying on complex WCET analysis during the development process. We believe that the development process is easier and faster with the stabilization framework. As we move to chip multiprocessors, high-performance real-time applications will have to be parallelized and predicting the execution time of CMP applications is even more difficult than for applications running on OoO processors. In the future we plan to develop a stabilization framework for CMPs.

References

[1] H. Aydin, V. Devadas and D. Zhu. System-level Energy Management for Periodic Real-Time Tasks. Proceedings of the 27th IEEE Real-Time Systems Symposium (RTSS'06), Rio de Janeiro, Brazil, December 2006.

[2] Benini, L., Bogliolo, A., and De Micheli, G. 2002. A survey of design techniques for system-level dynamic power management. In Readings in Hardware/Software Co-Design, G. De Micheli, R. Ernst, and W. Wolf, Eds. The Morgan Kaufmann Systems On Silicon Series. Kluwer Academic Publishers, Norwell, MA, 231-248.

[3] D. Brooks, V. Tiwari, and M. Martonosi. Wattch: A framework for architectural-level power analysis and optimizations. In 27th Annual International Symposium on Computer Architecture, June 2000.

[4] Doug Burger and Todd M. Austin. The SimpleScalar Tool Set Version 2.0. Technical Report 1342, Computer Sciences Department, University of Wisconsin--Madison, May 1997.

[5] Buce R. Childers, H. Tang and Rami Melhem, Adapting Processor Supply Voltage to Instruction-Level Parallelism, Koolchips 2000, during the 33rd Int'l. Symp. on Microarchitecture (MICRO-33), Monterey, CA, December 10, 2000.

[6] Cazorla, F. J., Knijnenburg, P. M., Sakellariou, R., Fernandez, E., Ramirez, A., and Valero, M. 2005. Architectural support for real-time task scheduling in SMT processors. In Proceedings of the 2005 international Conference on Compilers, Architectures and Synthesis For Embedded Systems (San Francisco, California, USA, September 24 - 27, 2005). CASES '05

[7] Cazorla, F. J., Knijnenburg, P. M., Sakellariou, R., Fernandez, E., Ramirez, A., and Valero, M. 2004. Predictable performance in SMT processors. In Proceedings of the 1st Conference on Computing Frontiers (Ischia, Italy, April 14 - 16, 2004). CF '04

[8] L. T. Clark, Circuit Design of XScale Microprocessors, In Proceedings of the 2001 Symposium on VLSI Circuits, June, 2001.

[9] Contreras, G., Martonosi, M., Peng, J., Lueh, G., and Ju, R. 2007. The XTREM power and performance simulator for the Intel XScale core: Design and experiences. Trans. on Embedded Computing Sys. 6, 1 (Feb. 2007), 4.

[10] Steven G. Dropsho, Greg Semeraro, David H. Albonesi, Grigorios Magklis, Michael L. Scott: Dynamically Trading Frequency for Complexity in a GALS Microprocessor. MICRO 2004: 157-168.

[11] K. Flautner, S. Reinhardt, and T. Mudge. Automatic Performance-Setting for Dynamic Voltage Scaling. In Proceedings of the 7th Annual international Conference on Mobile Computing and Networking (Rome, Italy). MobiCom '01. ACM, New York, NY, 260-271.

[12] Gene F. Franklin, J. David Powell, Abbas Emami-naeini, Feedback Control of Dynamic Systems, Addison-Wesley, 3rd Edition

[13] Guthaus, M. R., Ringenberg, J. S., Ernst, D., Austin, T. M., Mudge, T., and Brown, R. B. 2001. MiBench: A free, commercially representative embedded benchmark suite. In Proceedings of the Workload Characterization, 2001. Wwc-4. 2001 IEEE international Workshop on - Volume 00 (December 02 - 02, 2001).

[14] Greg Hamerly, Erez Perelman, Jeremy Lau, and Brad Calder, SimPoint 3.0: Faster and More Flexible Program Analysis, Journal of Instruction Level Parallelism, September 2005.

[15] A. Hergenhan and W. Rosenstiel. Static timing analysis of embedded software on advanced processor architectures. In Proceedings of Design, Automation and Test in Europe (DATE '00), pages 552--559, Paris, March 2000.

[16] C. J. Hughes et al. Variability in the Execution of Multimedia Applications and Implications for Architecture. In Proc. of the 28th Annual Intl. Symp. on Comp. Architecture, 2001.

[17] C. J. Hughes, J. Srinivasan, and S. V. Adve. Saving Energy with Architectural and Frequency Adaptations for Multimedia Applications. In Proceedings of the 34th Annual International Symposium on Microarchitecture (MICRO-34), Dec. 2001.

[18] Yoshifumi Ikenaga, Employ Supply Voltage Control to Save Energy, Energy-saving Designs, NEC corporation.
[19] Ishihara, T. and Yasuura, H. 1998. Voltage scheduling problem for dynamically variable voltage processors. In Proceedings of the 1998 international Symposium on Low Power Electronics and Design (Monterey, California, United States, August 10 - 12, 1998). ISLPED '98. ACM, New York, NY, 197-202.
[20] A. Iyer and D. Marculescu. Power and Performance Evaluation of Globally Asynchronous Locally Synchronous Processors. In Proceedings of the 29th International Symposium on Computer Architecture (ISCA), Anchorage, Alaska, May 2002.
[21] Jejurikar, R., Pereira, C., and Gupta, R. 2004. Leakage aware dynamic voltage scaling for real-time embedded systems. In Proceedings of the 41st Annual Conference on Design Automation (San Diego, CA, USA, June 07 - 11, 2004). DAC '04. ACM, New York, NY, 275-280
[22] Kondo, M., Sasaki, H., and Nakamura, H. 2007. Improving fairness, throughput and energy-efficiency on a chip multiprocessor through DVFS. SIGARCH Comput. Archit. News 35, 1 (Mar. 2007), 31-38.
[23] Y. Liu, H. Yang, R. P. Dick, H. Wang, and L. Shang, Thermal vs. energy optimization for DVFS-enabled processors in embedded systems, in Proc. IEEE Proc. Int. Symp. Quality Electronic Design, Jan. 2007.
[24] Z. Lu, J. Hein, M. Humphrey, M. Stan, J. Lach and K. Skadron. Control-theoretic dynamic frequency and voltage scaling. In Proceedings of the 2002 International Conference on Compilers, Architectures, and Synthesis for Embedded Systems (CASES), Grenoble, France, Oct. 2002.
[25] Mistry, K. Armstrong, M. Auth, C. Cea, S. Coan, T. Ghani, T. Hoffmann, T. Murthy, A. Sandford, J. Shaheed, R. Zawadzki, K. Zhang, K. Thompson, S. Bohr, M. Delaying forever: Uniaxial strained silicon transistors in a 90nm CMOS technology, Symposium on VLSI Technology, p. 50, (2004).
[26] Christian Poellabauer, Tao Zhang, Santosh Pande, and Karsten Schwan, An Efficient Frequency Scaling Approach for Energy-Aware Embedded Real-Time Systems, Proceedings of the International Conference on Architecture of Computing Systems (ARCS'05), Innsbruck, Austria, March 2005.
[27] E. Rotenberg. Using Variable-MHz Microprocessors to Efficiently Handle Uncertainty in Real-Time Systems. 34th International Symposium on Microarchitecture, December 2001.
[28] Semeraro, G.; Albonesi, D.H.; Magklis, G.; Scott, M.L.; Dropsho, S.G.; Dwarkadas, S., Hiding synchronization delays in a GALS processor microarchitecture, Asynchronous Circuits and Systems, 2004. Proceedings of the 10th International Symposium on Asynchronous Circuits and Systems (ASYNC'04).
[29] G. Semeraro, G. Magklis, R. Balasubramonian, D. H. Albonesi, S. Dwarkadas, and M. L. Scott, Dynamic frequency and voltage control for a multiple clock domain microarchitecture, in Intl. Symp. on Microarchitecture (MICRO), pp. 356-367, 2002.
[30] John Paul Shen, Lost in the Bermuda Triangle: Complexity vs Energy vs Performance, WCED, June 18, 2006. http://www.csl.cornell.edu/~albonesi/wced06/shen.pdf
[31] K. Skadron, T. Abdelzaher, and M. R. Stan. Control theoretic techniques and thermal-RC modeling for accurate and localized dynamic thermal management. Technical Report CS-2001-27.
[32] David C. Snowdon, Sergio Ruocco and Gernot Heiser, Power Management and Dynamic Voltage Scaling: Myths and Facts, Proceedings of the 2005 Workshop on Power Aware Real-time Computing, New Jersey, USA, September, 2005
[33] P. Stanley-Marbell and M. Hsiao and U. Kremer, A Hardware Architecture for Dynamic Performance and Energy Adaptation, In PACS-02, in conjunction with 8th IEEE International Symposium on High-Performance Computer Architecture.
[34] Y. Tan and V. Mooney, Timing Analysis for Preemptive Multi-tasking Real-Time Systems, Proceedings of Design, Automation and Test in Europe (DATE'04), February 2004.
[35] S.Thompson, M. Alavi, M. Hussein, P. Jacob, C. Kenyon, P. Moon, M. Prince, S. Sivakumar, S. Tyagi, and M. Bohr, "Semiconductor Technology and Manufacturing: 130nm Logic Technology Featuring 60nm Transistors, Low-K Dielectrics and Cu Interconnects," Intel Technology Journal, Volume 6, Issue 2, May 2002.
[36] Varma, A., Ganesh, B., Sen, M., Choudhury, S. R., Srinivasan, L., and Bruce, J. 2003. A control-theoretic approach to dynamic voltage scheduling. In Proceedings of the 2003 international Conference on Compilers, Architecture and Synthesis For Embedded Systems (San Jose, California, USA, October 30 - November 01, 2003). CASES '03. ACM, New York, NY, 255-266.
[37] C Xu, TM Le, TT Lay, H.264/AVC CODEC: Instruction Level Complexity Analysis. Ninth IASTED International Conference on Internet and Multimedia Systems and Applications; Honolulu, HI; USA; 15-17 Aug. 2005.
[38] Zhu, Y. and Mueller, F. Feedback EDF Scheduling Exploiting Dynamic Voltage Scaling. In Proceedings of the 10th IEEE Real-Time and Embedded Technology and Applications Symposium (Rtas'04) - Volume 00 (May 25 - 28, 2004).
[39] V. Zyuban and P. Strenski, Unified Methodology for Resolving Power-Performance Tradeoffs at the Microarchitectural and Circuit Levels, Proc. Int'l Symp. Low-Power Electronic Design (ISLPED 02), IEEE Press, 2002, pp. 166-171.